Chapter 11

MODELING SERVICE MIGRATION AND RELOCATION IN MISSION-CRITICAL SYSTEMS

Yanjun Zuo

Abstract Mission-critical information systems are commonly used in critical infrastructure assets such as the electric power grid, telecommunications networks, healthcare systems, water management systems and national defense. Damage or disruption of these systems could result in the loss of services and potentially serious societal consequences. Therefore, it is important to ensure that the essential services provided by these systems are reliable and dependable. This paper presents a modeling framework for service migration and relocation, which can dynamically transfer critical services from a compromised platform to other healthy platforms. The mechanism guarantees that vital services are continuously available despite malicious attacks and system failures. When the compromised platform has recovered, the services can be moved back to the platform. The modeling framework provides a means for studying the important factors that impact service migration and relocation, and how an assured service migration mechanism can be designed to increase confidence about the reliability of mission-critical information systems.

Keywords: Mission-critical systems, migration, reliability, availability

1. Introduction

Mission-critical systems provide vital services and must, therefore, be reliable and dependable despite malicious attacks and system failures. Their use in critical infrastructure assets such as the electric power grid, telecommunications networks, healthcare systems, water management systems and national defense raises serious concerns about their ability to withstand hardware and software failures, operator errors, power outages, environmental disasters and attacks by adversaries [9].

J. Butts and S. Shenoi (Eds.): Critical Infrastructure Protection VII, IFIP AICT 417, pp. 155–170, 2013.

Despite the best efforts of system developers and security practitioners, it is infeasible to assure that a mission-critical system will be invulnerable to well-organized attacks and unpredictable system failures. Indeed, the scale and complexity of mission-critical systems make it almost impossible to build completely secure and reliable systems. Consequently, it is prudent to prepare for the worst-case scenarios involving security incidents and system failures while ensuring that critical services continue to be available.

This paper focuses on service migration and relocation as a means for ensuring critical service availability. Informally, service migration is a process that suspends a critical service on its current faulty platform and moves its core programs and data to a clean, healthy platform where the service is resumed from where it left off. At a later time, after the compromised platform has recovered and the environment has been sanitized, the critical service is relocated to the original platform and its execution is resumed. Such migration and relocation is incorporated in the architectures of many safety-critical systems [6].

Compared with other survivability approaches, service migration is a viable solution to ensure that the most critical services are available in a challenging environment. Techniques such as fault tolerance/masking and damage avoidance are often too expensive or infeasible for large and complex mission-critical systems. In situations where faults are extensive and non-maskable, system operations cannot be continued, even when the system has multiple redundancies. Reconfiguration [7, 10] can avoid intensive replication, but it requires a change in the functional specifications of the platforms that are in operation. Dynamically reconfiguring a system is not always possible, especially when system components have been compromised and only untrustworthy components are available for reconfiguration. Unlike these techniques, strategic migration allows critical services to be continuously provided by other healthy platforms without incurring expensive resource duplication or attempting to reconfigure a system that is not trustworthy.

This paper presents a modeling framework for service migration and relocation in mission-critical systems. A stochastic process algebra, i.e., the Performance Evaluation Process Algebra (PEPA) [5], is used to represent the activities and components of a complete service migration and relocation procedure. Since critical services are essential, minimal time and effort should be required to move the mission-critical services and relocate them back after the original platform has recovered. This paper studies how important factors such as the probability of severe damage on a platform and the probability and duration of viable migration scheduling can impact the effectiveness and efficiency of service migration and relocation. The resulting model can be used to determine the important factors and how they impact service availability. Also, it can reduce the magnitude of service interruptions by helping deploy the most effective and efficient service migration and relocation mechanisms.

2. Related Work

Migration has been studied for a number of purposes, including application and service survivability, improvement of the quality of services, resource optimization, system agility and network virtualization. Migration has been proposed at a variety of levels, including services, system components, operating system processes, program threads and applications (e.g., mobile agents).

Choi, *et al.* [2] have proposed a methodology for run-time software component migration for application survivability in distributed real-time systems. Two properties are necessary for fast component migration: lightweight service and data transfer and proactive resource discovery. Choi and colleagues developed middleware to support prompt software component migration and identify the available resources to complete the migration. Experimental results demonstrate that the approach takes much less time compared with techniques based on reactive resource discovery.

Amoretti, *et al.* [1] have studied service migration in SP2A-based clouds (SP2A is a framework and middleware for peer-to-peer service-oriented architectures). They propose a framework and middleware for highly dynamic and adaptive clouds, characterized by peer providers and services that can be replicated by code mobility mechanisms.

Another service migration mechanism [4] moves the computational services of a virtual server to other available servers for adaptive grid computing. This enables computations to be resumed on a remote server without service reinstallation. The mechanism is incorporated in DSA, a Java compliant distributed virtual machine that accommodates adaptive parallel applications in grids.

Cohen, *et al.* [3] have proposed a service migration approach for enterprise system architectures. Instead of locating and delivering data to a processing site as in the case of traditional systems, services are delivered to the data sites for efficiency and higher levels of service availability. This approach extends the notion of a service-oriented architecture and is particularly effective when massive volumes of data (in terabytes) have to be processed. However, the approach requires new technical infrastructures and policies for client and server systems.

Li, *et al.* [8] have presented a service migration protocol that supports multimedia transfer in single-user, multiple-device scenarios. Data sessions are grouped by users and can seamlessly migrate to devices associated with the same user. A proxy is used to bridge a client and a server and a protocol is used to retain the current client and server operations while placing new functions at the proxy through naming and control and data plane designs.

The migration approaches described above are very specific and are defined at the lower level of system processes. Our methodology complements these approaches with the objective of providing a general analytic framework that can model and simulate service migration and relocation. Furthermore, it helps users identify the factors that influence the effectiveness and efficiency of service migration and relocation.

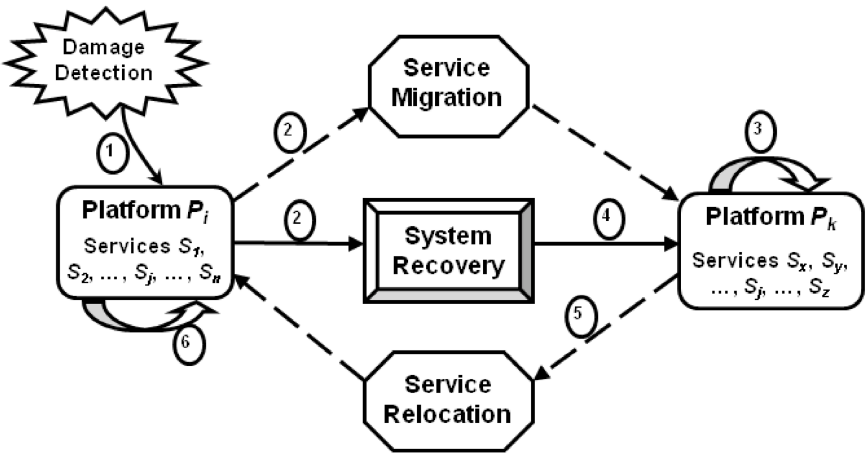

Figure 1. Workflow of a service migration and relocation process.

3. Service Migration and Relocation

Let SYS be a distributed system that provides critical services S_1, S_2, ..., S_n. SYS is composed of a set of computing platforms P_1, P_2, ..., P_m. Each service S_j is executed on one platform at any time and operates as a set of programs. It is assumed that service S_j is executed in a virtualized container and is, thus, self-contained, i.e., the service programs, data and processes can be referenced within its namespace. Technically, it is possible to move a service from its current platform P_i to a new platform P_k in a situation where P_i has been severely damaged. In such a case, allowing the platform to continue to perform mission-critical functions could be disastrous. To ensure that the critical services can be provided continuously and to avoid further losses, the services must be transferred to other healthy platforms that are immune to the same types of attacks that compromised the original platform.

Figure 1 shows the workflow of a service migration and relocation process. We assume that a set of services S_1, S_2, ..., S_n are currently executing on platform P_i. Suppose that abnormal behavior is detected on P_i and reported as shown in Step 1. This event triggers a thorough investigation and analysis of P_i by the security systems of SYS. If the damage assessment indicates that P_i has been severely damaged and service migration is the most appropriate strategy, then a service migration and relocation process initiates. For simplicity, we discuss the migration and relocation of one critical service S_j.

After a migration decision is made, two actions are taken concurrently as indicated by Step 2. The first action is to halt service S_j appropriately, e.g., by freezing the service processes, recording global data (service configuration and state), recording the states of individual processes and terminating the entire service program. As part of the second action, a migration scheduler attempts to generate a service migration arrangement for service S_j to be migrated from

P_i to a new, healthy platform P_k. The new platform must be able to support the core functions of S_j and should be immune to the same types of attacks suffered by P_i. Since a suitable platform may not always be available (e.g., in an environment in which resources are limited), we use a stochastic process to quantify the probability of whether or not an appropriate platform is available. If a platform is not available, the scheduler is modeled to constantly repeat the scheduling process – on the chance that a new platform becomes available at a later time when the operating environment has improved.

The service migration process is composed of three sub-processes [11]: (i) migration preparation, which saves the service programs and data in a resumeable image in a self-contained format with header, global data, internal process dependencies and shared resources (task structure and open files); (ii) service and data transfer, which synchronizes data copies, withdraws transactions, establishes recovery points and disseminates the packed service programs and data to the new platform P_k; and (iii) service setup on P_k, which creates a new namespace and restores the service configuration and state. Since some data associated with S_j may have been damaged, the system must provide supplemental data or generate fuzzy data to support the continuity of service S_j. When S_j is executed on the new platform P_k, the compromised platform P_i is immediately repaired. A recovery process in the model covers fault diagnosis and damage repair.

In Step 3 in Figure 1, after the service and data are set up on the new platform P_k, the execution of S_j is resumed. S_j may continue to execute on P_k to completion. However, for a long-running service, if the previously damaged platform P_i has recovered, then S_j must be moved back to P_i. This relocation is necessary for several reasons [11]: (i) it improves data access locality by relocating a service closer to the data; (ii) it provides better system response time by relocating the service closer to users; and (iii) it makes for better load balancing by relocating the service to its initial platform based on an optimal resource assignment scheme.

In Step 4, a repair completion notification is sent by the recovery manager to P_k when P_i is fully recovered. In Step 5, the relocation manager arranges to move S_j back to P_i. Service relocation is composed of three sub-actions similar to the migration process, but with one major difference – any fuzzy data used by S_j when it is running on P_k is meant to be used only temporarily while P_i is being repaired. A correctional recovery must be performed when S_j is relocated back to P_i so that the data items reflect the exact values instead of inaccurate, albeit acceptable, values. Finally, in Step 6, S_j runs to completion on the repaired platform P_i.

To ensure that critical services are provided with minimal interruption, we examine how important factors, such as various system security and functional properties, affect the activities that support an effective and efficient migration and relocation process. For example, the probability (or frequency) of severe damage to a platform significantly impacts the normal operations of the critical services on the platform. Intuitively, if a system has a strong security baseline,

few vulnerabilities, efficient intrusion detection and high levels of fault tolerance and masking abilities, then the probability of a platform being severely damaged would be low. Consequently, the critical services can operate on their original platforms most of the time. In such situations, the need for service migration and relocation is minimized. Even in a worst-case scenario, when a migration is necessary, it is necessary to ensure that some key system properties, if satisfied, will provide for effective and efficient service migration. For example, a new platform must be available to host the services when damage is detected to the original platform. Halting the services on the compromised platform and setting them up on the new platform must be completed as quickly as possible. The service priority, response time and throughput must be ensured both during and after the service migration. As will be seen below, these factors are expressed as model parameters (e.g., activity rates) in a simulation because we are interested in learning how the parameters affect the normal execution of critical services.

4. Service Migration and Relocation Modeling

We use the PEPA model [5] to express and simulate the activities and behavior of system components in a service migration and relocation process. PEPA introduces delays and probabilistic occurrences to process algebras. The timing behavior of a system is quantified by associating a random variable with each activity, which represents its duration. Behavior uncertainty is determined by probabilistic branching – the probabilities of occurrence of some activities are determined by race conditions between the enabled activities. The model represents service migration and relocation activities as stochastic actions that are non-deterministic and whose occurrence or non-occurrence are predicted by one or more random variables (i.e., activity rates).

Figure 2 shows the PEPA model representing a system SYS. The system SYS is modeled in terms of interactions between the service migration and relocation components (i.e., a migrating scheduler, a platform supporting a critical service and a relocation manager) and the damage recovery components (i.e., a fault diagnosing agent and a damage repairer). The model has eleven processes (components) representing the entire procedure for service S_j to migrate from a compromised platform P_i to a new platform P_k and relocate from P_k to P_i after P_i has recovered. The model also includes the recovery procedure of the compromised component. The PEPA processes and their activities are described below.

Process $Execution_{i\_j}$ models the behavior of platform P_i where service S_j is executed. It either executes normally or needs to be investigated if the intrusion detection system (implicitly modeled) reports suspicious (or abnormal) behavior. Correspondingly, P_i behaves either as $(monitor\_anomaly, p_1).(alarm, al)$ $.Contingency_i$ or $(monitor\_normal, p_2).(executing_{i\_j}, f).Execution_{i\_j}$ in the PEPA model. The former describes the situation where the system reports damage symptoms (represented by the PEPA activity $monitor\_anomaly$). In this case, an alarm is trigged (represented by the PEPA activity $alarm$) and P_i enters a contingency state (represented by the PEPA process $Contingency_i$),

$Execution_{i\_j} \equiv (monitor\_anomaly, p_1).(alarm, al).Contingency_i + (monitor\_normal, p_2).(executing_{i\_j}, f).Execution_{i\_j};$

$Contingency_i \equiv (investigate\_damaged, (it * p_3)).(recovery\_notify_i, tt).Migration\_Manager + (investigate\_self\_contain, (it * p_4)).Execution_{i\_j};$

$Migration\_Manager \equiv (halting_{i\_j}, h).Migration\_Scheduler;$

$Migration\_Schedule \equiv (schedule\_ok,(st * p_5)).Migration_{i\_k} + (schedule\_failure,(st * p_6)).Migration\_Scheduler;$

$Migration_{i\_k} \equiv (m\_Pre, m_1).(m\_Tr, m_2).(m\_Su, m_3).Execution_{k\_j};$

$Execution_{k\_j} \equiv (recovery\_check\_ok, p_7).(recovered_i, T).Relocation\_Manager + (recovery\_check\_pending, p_8).(executing_{k\_j}, f).Execution_{k\_j};$

$Relocation\_Manager \equiv (halting_{k\_j}, h).Relocation_{k\_i};$

$Relocation_{k\_i} \equiv (r\_Pre, r_1).(r\_Tr, r_2).(r\_Su, r_3).Execution_{i\_j};$

$Recovery\_Manager \equiv (recovery\_notify_i, T).Recovery_i;$

$Recovery_i \equiv (diagnose, dt).Repairer;$

$Repairer \equiv (repair\_success, (l * p_9)).(recovered_i, rp).Recovery\_Manager + (repair\_fail, (l * p_{10})).Recovery_i;$

$SYS \equiv Execution_{i\_j} \underset{L}{\bowtie} Recovery\_Manager$

$(L = \{recovery\_notify_i, recovered_i\})$

Figure 2. PEPA model of service migration and relocation.

where the symptoms are further analyzed. The latter represents the case when no abnormal symptom is detected (represented by the PEPA activity $monitor\_normal$). Therefore, S_j is continuously executed on P_i (represented by $executing_{i\_j}$). The probabilities of occurrence of the two cases are denoted by p_1 and p_2, respectively. In our model, $monitor\_anomaly$ and $monitor\_normal$ have default activity rates of one; $alarm$ and $executing_{i\_j}$ have the rates al and f, respectively, indicating that the two activities have durations that are negatively exponentially distributed with parameters al and f, respectively. The probabilities that $alarm$ and $executing_{i\_j}$ occur within a period of time length t are $1 - e^{-al*t}$ and $1 - e^{-f*t}$, respectively.

The PEPA process $Contingency_i$ represents the investigation of platform P_i for advanced analysis given the reported suspicious behavior. Based on the investigation results, $Contingency_i$ behaves either as $(investigate\_damaged, (it * p_3)).(recovery\_notify_i, tt).Migration\_Manager$ or $(investigate\_self\_contain, (it * p_4)).Execution_{i\_j}$. In the first case, the investigation reveals that P_i is severely damaged (represented by the PEPA activity $investigate\_damaged$). Thus, a recovery notification ($recovery\_notify_i$) is sent to $Recovery\_Manager$ in order to arrange for P_i to recover. Furthermore, the system starts the service migration process (represented by the PEPA process $Migration\_Manager$) to

move the critical service S_j executing on P_i to a new, healthy platform. In the second case, the damage is not severe and the system can self-contain or mask the damage without disrupting S_j (represented by $investigate\_self\_contain$). Therefore, the workflow transits back to the normal execution of S_j on P_i, i.e., $Execution_{i\_j}$. The probabilities of the two investigation results are denoted by p_3 and p_4, respectively. The activity rate it probabilistically determines the duration of the investigation procedure, i.e., the probability that the investigation is completed within a period of time t is $1 - e^{-it*t}$.

Service migration is modeled using three processes: $Migration\_Manager$, $Migration\_Scheduler$ and $Migration_{i\_k}$. $Migration\_Manager$ initializes the migration procedure by halting service S_j on P_i (represented by the PEPA activity $halting_{i\_j}$ with a rate h). $Migration\_Scheduler$ represents the behavior of the migration scheduler, which identifies a new platform P_k, if one exists, and arranges for S_j to migrate to P_k. The scheduling process takes an amount of time that is negatively exponentially distributed with parameter st. As discussed earlier, the new platform must be capable of providing the functions required by S_j and in the meantime be immune to the same types of attacks that compromised platform P_i.

The scheduling process yields one of two possible results: (i) a suitable platform P_k is identified (represented by the PEPA activity $schedule\_ok$) with probability p_5; and (ii) no P_k is available given the current system resources (represented by $schedule\_failure$) with probability p_6. For the first case, the actual migration can start as represented by the PEPA process $Migration_{i\_k}$. $Migration_{i\_k}$ engages three activities m_Pre, m_Tr and m_Su, which correspond to migration preparation, service and data transfer, and setup on the new platform P_k, respectively. Their corresponding PEPA activity rates are m_1, m_2 and m_3. However, if the scheduling process does not identify a suitable P_k, then the workflow returns to $Migration\_Scheduler$ for rescheduling (hopefully a suitable platform will be available the next time).

The PEPA process $Execution_{k\_j}$ represents the execution of service S_j on the new platform P_k. Depending on whether the compromised platform P_i has recovered or not, P_k behaves differently. As discussed above, S_j has to be relocated when P_i has recovered. The first behavior of the process $Execution_{k\_j}$, i.e., $(recovery\_check\_ok, p_7).(recovered_i, T).Relocation\_Manager$ covers this case with a probability p_7 when the status of P_i is periodically checked ($recover\ y\_check\_ok$). Indeed, in this case, P_k synchronizes with the $Recovery\_Manager$ through the activity $recovered_i$. This activity has an unspecified rate (denoted by T and is determined by the repair notification rate rp when the two processes $Execution_{k\_j}$ and $Recovery\_Manager$ are synchronized). Then, the system starts the relocation process $Relocation_{k\_i}$, which moves S_j to its original platform P_i. $Relocation_{k\_i}$ engages three activities r_Pre, r_Tr and r_Su. The second behavior of $Execution_{k\_j}$, i.e., $(recovery\_check\_pending, p_8).(executing_{k\_j}, f).Execution_{k\_j}$ represents the continuous execution of S_j on P_k given the condition that P_i is still being repaired (represented by the PEPA activity $recovery\_check\_pending$). The probability of this case is represented by p_8. In

terms of time, S_j is executed on P_k (represented by $executing_{k\_j}$) for a duration that is negatively exponentially distributed with a parameter f before the process returns to $Execution_{k\_j}$ for the next cycle of repair checking.

The damage recovery subsystem of SYS is modeled by three PEPA components: (i) $Recovery\_Manager$, a coordinator that waits for a recovery notification (represented by $recovery\_notify_i$) and then starts a recovery process; (ii) $Recovery_i$, a local agent responsible for recovering platform P_i; it first performs fault diagnosis on P_i (represented by the PEPA activity *diagnose* with a rate *dt*) and then develops a plan for repair; and (iii) *Repairer*, a component responsible for the actual repair of P_i. System repair is application-specific and can take various forms, e.g., system restoration, check pointing and rollback, and re-programming. Given the diagnosis information, P_i may be successfully repaired (represented by $repair\_success$) with probability p_9 or repair is not possible (represented by the activity $repair\_fail$) with probability p_{10}. In the first case, the workflow returns to the PEPA process $Recovery\_Manager$, indicating that the damage recovery system has completed the recovery of P_i and is waiting for the next recovery request. In the second case, since the repair attempt has not resulted in successful repair, further diagnosis is necessary. Therefore, the workflow returns to $Recovery_i$ for more diagnostic information about the cause and nature of the damage on the promised platform.

Finally, the service migration and relocation enabled system SYS is modeled using the PEPA cooperation operator $Execution_{i\_j} \bowtie_L Recovery\_Manager$ (where $L = \{recovery\_notify_i, recovered_i\}$) to represent the concurrent interactions of the execution of service S_j on platform P_i, and the damage recovery activities in the case of severe damage to P_i, via synchronized participation in the events $Recovery\_notify_i$ and $recovered_i$. While a severe damage event triggers the service migration process, a successful repair results in the migrated service being moved back to the original platform.

5. Simulation Results and Analysis

The PEPA model presented in this paper was solved using the PEPA Eclipse plug-in software [5]. The activity rates used in the model are shown in Table 1. A Bayesian network decision model was used to determine the activity rates. Due to space limitations, the Bayesian network model is not presented in this paper.

5.1 Overview

We conducted several rounds of simulations for steady-state, utilization, passage-time, throughput and experimental analysis in order to study how the important factors influence the effectiveness and efficiency of service migration and relocation.

In order to run the PEPA simulation, the system states corresponding to the underlying continuous time Markov processes were derived and the probability of the system at each state was generated. The PEPA model incorporated 33

Table 1. Parameter settings for the PIPA model.

Parameter	Value	Explanation
p_1	0.001	Probability that an anomaly is detected on platform P_i
al	0.5	Alarm on P_i is fired for two time units
p_2	$1 - p_1$	Probability that the intrusion detection system reports normal activities
it	0.1	Investigation of P_i takes ten time units upon triggering an alarm
f	0.1	S_j is executed on P_i for ten time units before the next intrusion report
p_3	0.1	Probability that P_i is severely damaged
tt	1	Repair notification to the $Recovery\_Manager$ takes one time unit
p_4	$1 - p_3$	Probability that the damage is not severe and P_i can contain the damage
h	0.2	Service S_j takes five time units to be halted appropriately
st	0.3	Migration scheduling takes about 3.3 time units
p_5	0.8	Probability that a new platform P_k is identified for S_j to migrate
p_6	$1 - p_5$	Probability that no suitable platform is available for S_j to migrate
m_1	0.25	Migration preparation takes four time units
m_2	0.5	Data and service transfer to the new platform P_k takes two time units
m_3	0.2	Setting up service S_j on P_k takes five time units
p_7	0.01	Probability that P_i has recovered while S_j is executing on P_k
p_8	$1 - p_7$	Probability that P_i still has to recover
r_1	0.25	Relocation preparation takes four time units
r_2	0.5	Data and service transfer back to the repaired P_i takes two time units
r_3	0.1	Setting up service S_j on P_i takes five time units
dt	0.1	Diagnosing the nature of the damage to P_i takes ten time units
l	0.001	Attempting to repair P_i takes 1,000 time units
p_9	0.6	Probability that P_i can be successfully repaired
p_10	$1 - p_9$	Probability that P_i cannot be repaired (new diagnosis is necessary)
rp	1	Sending a repair completion notification to P_k takes one time unit

global states. Since the model had two top-level PEPA processes, $Execution_{i\_j}$ and $Recovery\_Manager$, a global state had two elements, one for each local state of the corresponding top-level processes. The simulation revealed that

$Execution_{i\_j}$ had seventeen local states while $Recovery\_Manager$ had four states.

The system activities were represented as non-deterministic stochastic actions whose occurrence or non-occurrence were predicted by one or more random variables. The system SYS was modeled in terms of interactions between the service migration and relocation components and the damage recovery subsystems. The interactions reached a set of steady states after an extended execution time. The term "steady state" means that there was a statistically determined possibility that the system remained in the state.

Our simulations showed that the PEPA states with dominating steady-state probabilities were associated with the two local states $executing_{i\_j}.Execution_{i\_j}$ (0.891) and $Recovery\_Manager$ (0.981). This was also observed in our utilization analysis, which showed the long-run utilization of each top-level process in the PEPA model. More precisely, the percentage of time that a top-level component $Execution_{i\_j}$ (resp. $Recovery\_Manager$) was in the local state $executing_{i\_j}.Execution_{i\_j}$ (resp. $Recovery\_Manager$) was 0.891 (resp. 0.981). Since $executing_{i\_j}.Execution_{i\_j}$ represents the normal execution of service S_j on its original platform P_i, maximizing the utilization of this state is the objective of an efficient service migration and relocation process. Therefore, in the following discussion, we focus on how the important factors influence the utilization of $executing_{i\_j}.Execution_{i\_j}$. This utilization rate is denoted by p.

The activity throughput was also studied in the simulations. The throughput analysis yielded the rate at which PEPA actions were performed at steady-state. The two PEPA activities with the highest throughputs were $executing_{i\_j}$ (0.09) and *monitoring_normal* (0.09). The former indicates that service S_j was executing on platform P_i and, hence, a higher value is more desirable. The latter indicates that the intrusion detection system reported normal system operations in most cases (i.e., no suspicious behavior was detected). Just as the steady-state analysis focused on the local state $executing_{i\_j}.Execution_{i\_j}$, the throughput of $executing_{i\_j}$ represented the desired behavior of S_j on P_i; therefore, it is another metric of interest.

5.2 Simulation Results

We executed the PEPA model with parameter values in the desired ranges. The experiments were designed to study how system security and functional factors affect the utilization rate of $executing_{i\_j}.Execution_{i\_j}$, i.e., p and the throughput of $executing_{i\_j}$ as mentioned above.

We started with the two factors determined by the security features of system SYS: (i) probability of anomaly detection on a platform P_i (p_1) in an intrusion detection reporting cycle; and (ii) probability that the detected damage is severe (p_3). Intuitively, if a system has strong security mechanisms and the ability to contain and mask potential damage, then the need for the critical services to migrate from their normal executing platforms is low. Hence, the utilization of $executing_{i\_j}.Execution_{i\_j}$ should be higher. The experimental results confirmed this observation.

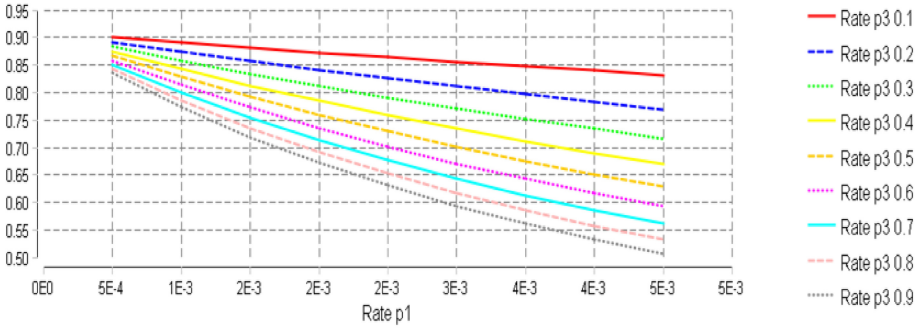

Figure 3. Utilization of $executing_{i\_j}.Execution_{i\_j}$ (p_1 and p_3 values).

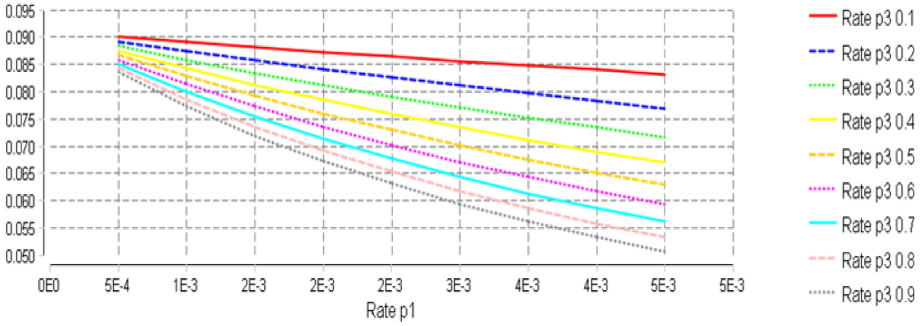

Figure 4. Throughput of activity $executing_{i\_j}$ (p_1 and p_3 values).

Figure 3 shows the utilization of $executing_{i\_j}.Execution_{i\_j}$ for different p_1 and p_3 values. In particular, it shows that p decreases when p_1 and p_3 increase. This clearly indicates that a higher compromise rate for a platform decreases the amount of time that the platform effectively supports the critical services. Furthermore, the quantitative relationship between p and p_1 is roughly linear given a fixed p_3 rate. This implies that a significant improvement in system security results in an almost equal increase in the normal execution of critical services on their original platforms. A similar pattern is observed in Figure 4 for the throughput of $executing_{i\_j}$ for different p_1 and p_3 values.

As discussed earlier, the migration manager is responsible for halting a critical service on a compromised platform, scheduling and arranging a new platform for service migration, moving the data and program space of the service to the new platform, and finally setting up the service on the new platform. In the meantime, the recovery manager diagnoses the faults and attempts to repair the compromised platform. The performance of these two system components clearly affects service migration. Figures 5 and 6 show that a higher probability of a successful migration-scheduling rate (i.e., p_5) and a higher probability of a successful repair of a compromised platform (i.e., p_9) positively affect the utilization of $executing_{i\_j}.Execution_{i\_j}$ (i.e., p). This indicates that effective damage recovery and the availability of healthy platforms increase the overall

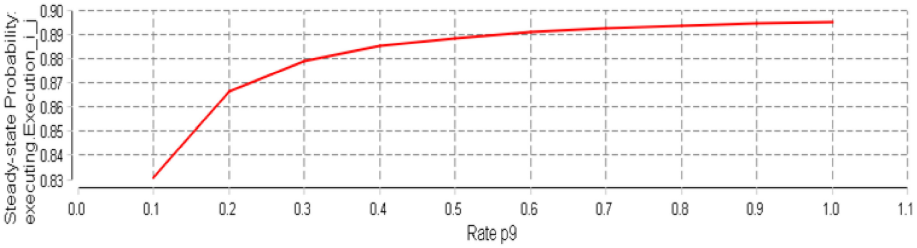

Figure 5. Utilization of $executing_{i\_j}.Execution_{i\_j}$ (p_5 values).

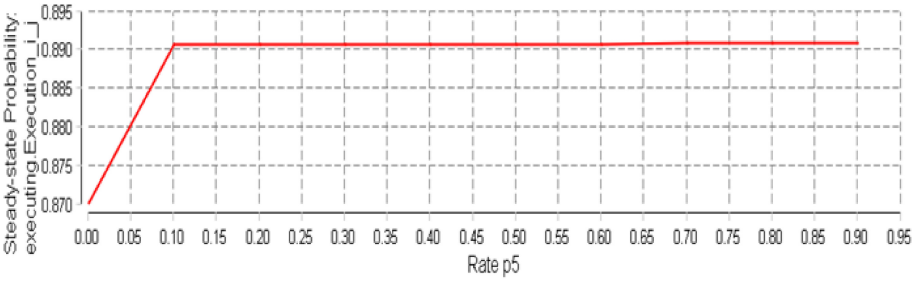

Figure 6. Utilization of $executing_{i\_j}.Execution_{i\_j}$ (p_9 values).

efficiency of service migration, which, in turn, increases the percentage of time that critical services are executed on their normal platforms. However, Figure 6 shows that p becomes stable after p_5 reaches a certain value (0.1 in our simulation). This means that any further improvement in migration scheduling will not improve p significantly. Therefore, migration scheduling is not a significant bottleneck for $executing_{i\_j}.Execution_{i\_j}$ beyond this point.

Next, we examine how the time required to halt a critical service on a compromised platform and the time required to repair the compromised platform affect the utilization of $executing_{i\_j}.Execution_{i\_j}$. Figure 7 shows that when less time is taken to repair a compromised platform (i.e., higher value of l), the probability that S_j is executed on platform P_i is higher. Similarly, the simulation shows that a shorter duration for halting S_j on P_i (i.e., higher value of h) and a shorter duration for migration scheduling to move S_j to a new platform P_k (i.e., higher value of st) both result in a higher value of p. However, both results indicate that the effects of h and l on p are not significant beyond certain points. This implies that further investments in halting and repairing mechanisms may not significantly increase the probability that critical services are executed on their normal platforms.

5.3 Passage-Time Analysis

We also conducted various passage-time analyses, i.e., the distribution of the probabilities that a second event occurs after a given event within a time duration. Figure 8 shows the passage-time analysis results of detecting abnormal symptoms ($monitor\_abnormal$) relative to service halting ($halting_i$).

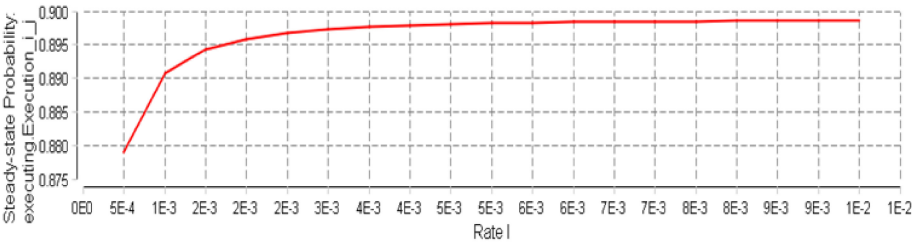

Figure 7. Utilization of $executing_{i\_j}.Execution_{i\_j}$ (l values).

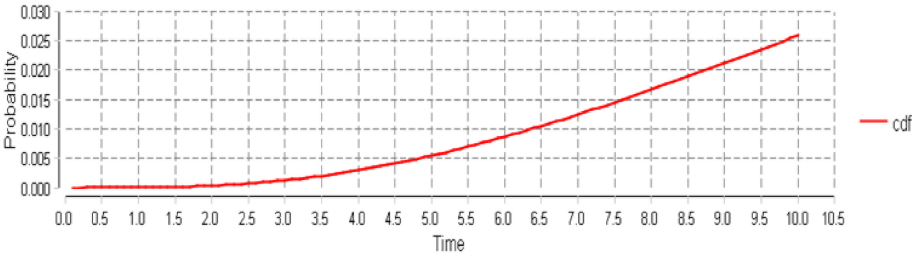

Figure 8. Detection of abnormal symptoms relative to service halting.

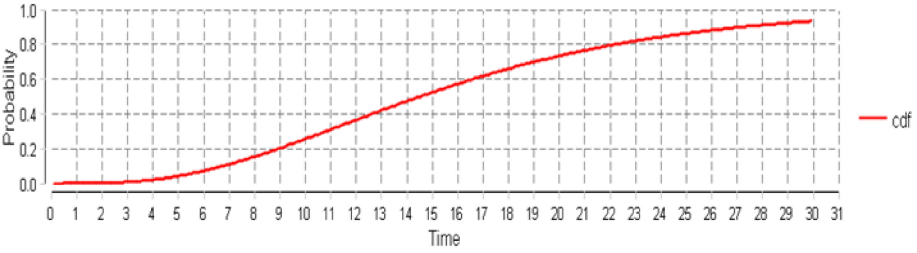

Figure 9. Service setup on a new platform relative to completion.

The results reveal that the occurrence probability of halting service S_j on P_i (i.e., $halting_i$) increases with time relative to the event where P_i is detected as damaged.

The passage-time analysis results in Figure 9 show that the probability of P_i being repaired (i.e., $recovered_i$) increases with time relative to the event that S_j executes on the new platform P_k (i.e., SU_{k_j}). In particular, it is almost certain that the compromised platform P_i will be repaired within about 30 time units after SU_{k_j}.

6. Conclusions

Service migration and relocation is an effective mechanism for dynamically transferring mission-critical services from a compromised platform to other clean, healthy platforms to ensure that mission-critical services are provided continuously. Service migration is crucial in situations where other security and fault tolerance approaches are infeasible or expensive. In these situations,

service migration and relocation is a viable solution for minimizing the impact of malicious attacks and system failures. The process-algebra-based modeling framework presented in this paper provides a foundation for studying how important factors can influence the effectiveness and efficiency of service migration and relocation in mission-critical systems.

Acknowledgement

This research was supported by the U.S. Air Force Office of Scientific Research under Award No. FA9550-12-1-0131.

References

[1] M. Amoretti, M. Laghi, F. Tassoni and F. Zanichelli, Service migration within the cloud: Code mobility in SP2A, *Proceedings of the International Conference on High Performance Computing and Simulation*, pp. 196–202, 2010.

[2] B. Choi, S. Rho and R. Bettati, Fast software component migration for application survivability in distributed real-time systems, *Proceedings of the Seventh IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pp. 269–276, 2004.

[3] S. Cohen, W. Money and S. Kaisler, Service migration in an enterprise system architecture, *Proceedings of the Forty-Second Hawaii International Conference on System Sciences*, 2009.

[4] S. Fu and C. Xu, Service migration in distributed virtual machines for adaptive grid computing, *Proceedings of the International Conference on Parallel Processing*, pp. 358–365, 2005.

[5] J. Hillston and S. Gilmore, Performance Evaluation Process Algebra (PEPA), Laboratory for Foundations in Computer Science, University of Edinburgh, Edinburgh, United Kingdom (`www.dcs.ed.ac.uk/pepa`), 2012.

[6] A. Keromytis, J. Parekh, P. Gross, G. Kaiser, V. Mishra, J. Nieh, D. Rubenstein and S. Stolfo, A holistic approach to service survivability, *Proceedings of the First ACM Workshop on Survivable and Self-Regenerative Systems*, pp. 11–22, 2003.

[7] M. Koester, W. Luk, J. Hagemeyer, W. Porrmann and U. Ruckert, Design optimizations for tiled partially reconfigurable systems, *IEEE Transactions on Very Large Scale Integration Systems*, vol. 19(6), pp. 1048–1061, 2011.

[8] C. Li, I. Pefkianakis, B. Li, C. Peng, W. Zhang and S. Lu, A multimedia service migration protocol for single user multiple devices, *Proceedings of the IEEE International Conference on Communications*, pp. 1923–1927, 2012.

[9] R. Marsh, Critical Foundations: Protecting America's Infrastructures, Report of the President's Commission on Critical Infrastructure Protection, United States Government Printing Office, Washington, DC, 1997.

[10] K. Zick and J. Hayes, Low-cost sensing with ring oscillator arrays for healthier reconfigurable systems, *ACM Transactions on Reconfigurable Technology and Systems*, vol. 5(1), article no. 1, pp. 1.1–1.26, 2012.

[11] Y. Zuo, Moving and relocating: A logical framework of service migration for software system survivability, *Proceedings of the Seventh IEEE International Conference on Software System Survivability*, p. 10, 2013.